Fondements mathématiques de l'Informatique (INF 1051)

Etienne Kouokam

UMMISCO-LIRIMA, Département d'Informatique
Université de Yaoundé I, Cameroun

Année académique 2019-2020

Contenu

Objectifs

Objectifs

Ce cours s'adresse aux étudiants de licence info et participe à leur faire découvrir :

- Les systèmes de numération : premiers pas vers la notion de représentation finie informations en machine
- Certaines bases mathématiques :
  - Systèmes formels
  - Logique des propositions
  - Logique des prédicats du premier ordre
- L'utilisation de la déduction en programmation logique qui combine :
  - La nature déclarative de la logique elle-même
  - Les techniques de programmation nécessaires
- Comment mener des preuves d'algorithme

Ressources bibliographiques

Les livres

Prérequis : Eléments d'Algèbre

Ouvrages :

1. Méthodes mathématiques pour l'informatique, Jacques Vélu. DUNOD
2. Y. N. Sifgh ; Mathematical Foundation of Computer Science ; New Age International (P) Ltd., Publishers ; New Delhi ; 2005
3. First-Order Logic and Automated Theorem Proving, Melvin Fitting, Springer-Verlag Telos, 1990. International, 1995
4. Logic and Structure, Dirk van Dalen, Springer, 2004.
5. Michael Spivey, "An introduction to logic programming through Prolog", Prentice-Hall

Support de cours : Conception en cours . . .

Plan

Informatique : What is it ?

INFORMATIQUE= INFORMATION + AUTOMATIQUE

- Science du traitement automatique et rationnel de l'information considérée comme le support des connaissances et des communications.
- Domaine d'activité scientifique, technique et industriel concernant le traitement automatique de l'information par l'exécution de programmes informatiques par des machines : systèmes embarqués, ordinateurs, robots, automates, ...

Pour quoi faire ?

- Représenter, Modéliser
- Approximer
- Calculer sur des modèles, des représentations
- Résoudre de manière efficace et précise (et le prouver)

Un exemple : prévision météorologique

Un problème : prévoir le temps

Etape 1 : Acquisition des données

- nombreuses données
- sources très différentes : images satellite, relevés au sol et par ballon sonde, radar météorologique

Etape 2 : Modèle numérique

- équations qui régissent l'atmosphère (mécanique des fluides)
- impossible de tout prendre en compte ! approximation

Etape 3 : Simulation

- Calculer pas à pas l'évolution du système suivant le modèle numérique
- De nos jours, plusieurs simulations sont lancées en parallèle avec des modèles différents

Inventions & Théories

- La machine à calculer de PASCAL (1623-1662) : effectue mécaniquement additions et soustractions.
- LEIBNIZ (1646-1716) : machine qui raisonne → enchaîne des propositions élémentaires pour effectuer des déductions. Rajout de la multiplication et de la division à la machine de PASCAL.
- CHARLES BABBAGE (1792-1871) construit en 1833 une machine à calculer, la machine à différences.
- En 1854, un mathématicien anglais, GEORGE BOOLE (1815-1864), présente une algèbre pour simuler les raisonnements logiques.
- CLAUDE SHANNON (1916-2001) fit le rapprochement entre les nombres binaires, l'algèbre de Boole et les circuits électriques. Il montra que le système binaire, manipulé par les opérations logiques de BOOLE (en prenant 1=vrai et 0=faux), permettait de réaliser toutes les opérations logiques et arithmétiques.

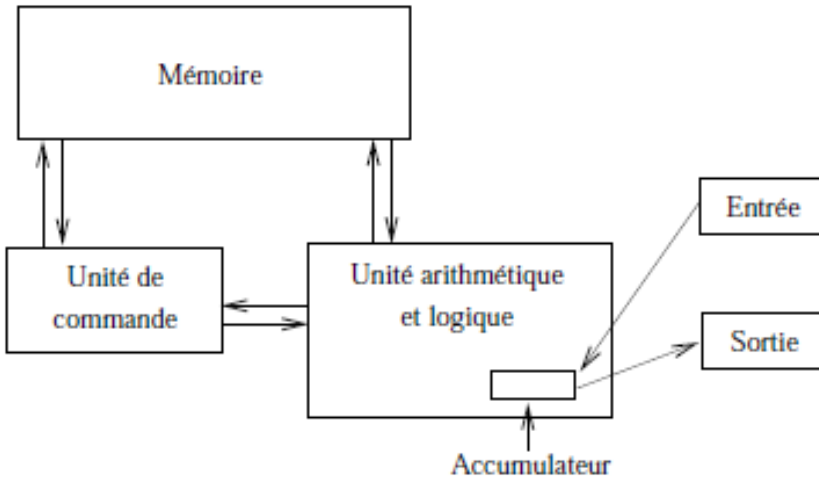
Naissance de l'ordinateur

1944 : un calculateur (le Mark1 d'IBM) pouvait multiplier 2 nombres de 23 chiffres en 6 secondes.

1945 : ENIAC (Electronic Numerical Integrator And Calculator) 18 000 tubes, 30 tonnes, multiplie 2 nombres de 10 chiffres en 3 millisecondes. Inconvénients : les données sont sur des cartes perforées mais les programmes sont câblés en mémoire et donc, pour passer d'un programme à un autre, il faut débrancher et rebrancher des centaines de câbles (ce n'est pas vraiment le premier ordinateur au sens actuel).

Fin 1945 : JOHN VON NEUMANN, associé à l'ENIAC, propose un modèle d'ordinateur qui fait abstraction du programme et se lance dans la construction d'un EDVAC (Electronic Discrete Variable Automatic Computer) : l'IAS. L'ordinateur est né, c'est la **machine de VON NEUMANN** décrivant les cinq composants essentiels de ce qu'on appellera l'architecture de VON NEUMANN

- Unité arithmétique et logique (UAL)
- Unité de commande ;
- Mémoire centrale
- Unité d'entrée
- Unité de sortie



Caractéristiques de la machine de Von Neumann

- Machine universelle contrôlée par programme
- Instructions et données, sous format binaire, stockées en mémoire (dans la même mémoire) ;
- Le programme peut modifier ses propres instructions ;
- Ruptures de séquence.

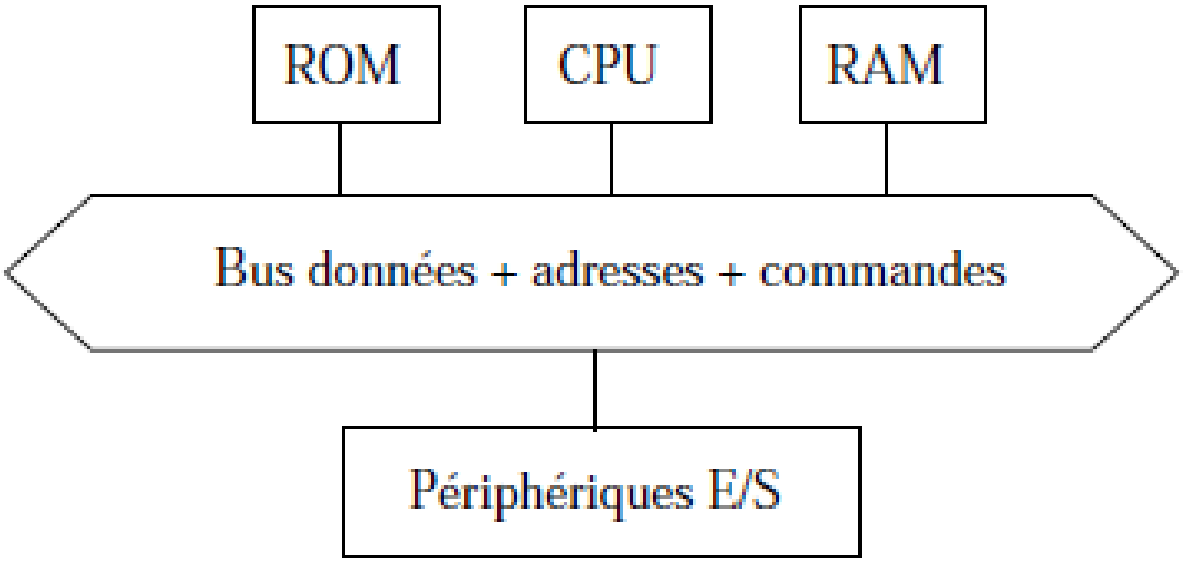
Caractéristiques de la machine de Von Neumann

1ère génération (1945-1955) : tubes à vide (pannes fréquentes, difficiles à déceler, ordinateurs demandant beaucoup de place), et tores de ferrite pour les mémoires.

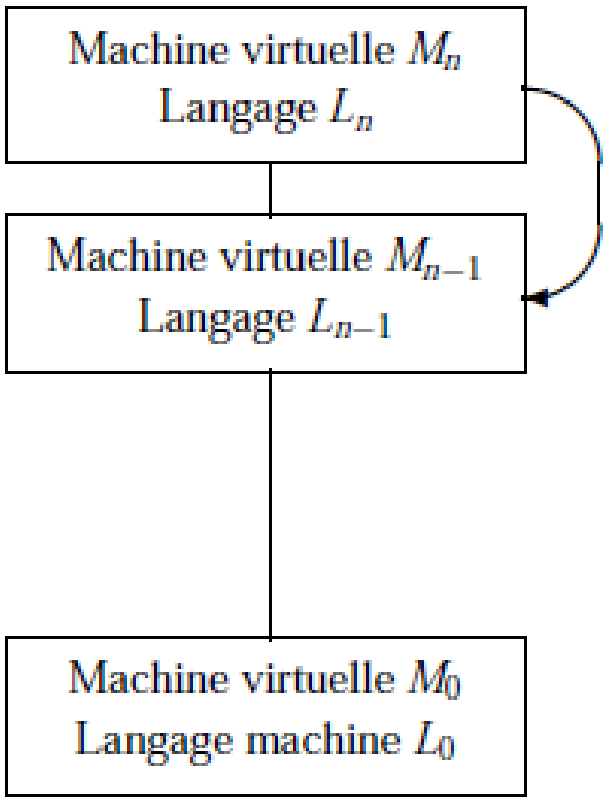
2ème génération (1955-1965) : remplacement des tubes par des transistors ; organisation de la machine autour d'un bus ; stockage sur bande magnétique ; écrans ; . . .

3ème génération (1965-1980) : apparition des circuits intégrés (puces) et des processeurs) éloignement du schéma de VON NEUMANN car processeurs d'E/S pour traitement direct avec la mémoire ; miniaturisation (ordinateurs plus petits, plus rapides et moins chers) ; gammes de machines (même langage d'assemblage pour des machines différentes, d'où réutilisabilité des programmes) ; multiprogrammation (plusieurs programmes en mémoire, lorsqu'un programme entre en phase d'entrées-sorties, l'UC passe à l'exécution d'une partie d'un autre programme).

Structure générale d'un ordinateur mono-processeur actuel



- L'activité d'une machine est rythmée par le CPU qui exécute le cycle suivant : chercher une instruction en mémoire (RAM), l'exécuter, chercher l'instruction suivante en mémoire, ...
- Une instruction est une suite de bits contenant le code de l'opération (addition, soustraction, recherche d'un mot en mémoire, ...) et ses opérandes (arguments de l'opération).
- Pour éviter d'avoir à connaître ce langage difficilement manipulable par l'homme, on définit des langages de plus haut niveau d'où une architecture en couches de machines virtuelles



Les machines actuelles présentent généralement l'architecture en couche ci-dessous :

Niveau 5	**Langages d'application**		Pascal, C, Cobol, Basic, Prolog, Caml, Lisp, Java
	↓ ↓	interprétation ou compilation	
Niveau 4	**Langage d'assemblage**		Instructions assembleur, notion de variables, de piles d'exécution, de routines
	↓	traduction assembleur	
Niveau 3	**Système d'exploitation**		Routines programmées
	↓	interprétation	
Niveau 2	**Couche machine**		Langage machine codé en binaire
	↓ ↓	interprétation (microprogramme)	
Niveau 1	**Couche microprogrammée**		Langage interne du microprocesseur
	↓ ↓	exécution directe par le matériel	

Information & Codage

- Processeur = système automatique de traitement d'information
- Information = éléments tels que texte, parole, image, mesure d'une grandeur physique, nombre, ...
- Information représentée sous une forme physique appropriée au traitement qu'elle doit subir
- Première étape essentielle : codage de l'information. Signaux (images, paroles, textes) codés in fine sous forme de 0 et de 1 (système binaire).

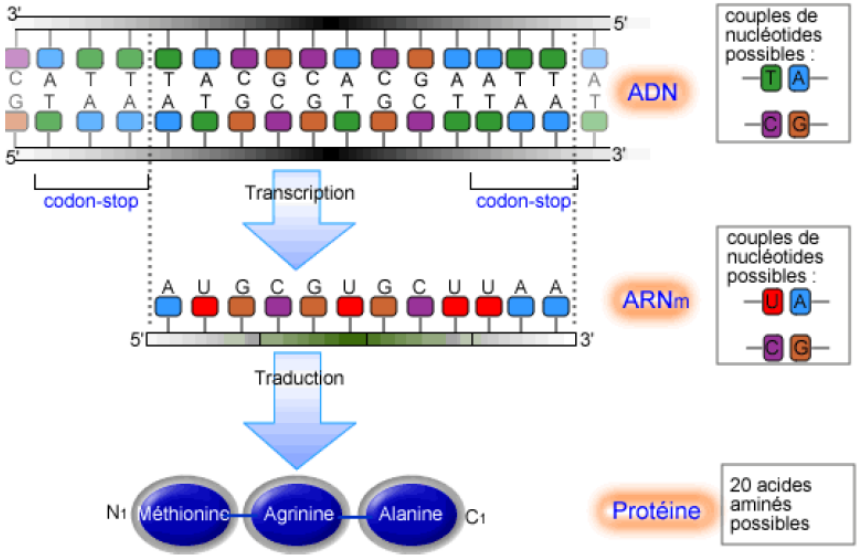
De l'information au bit

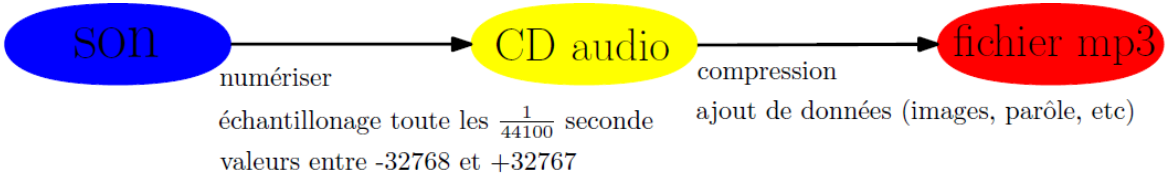
Nombreuses étapes entre le phénomène réel et son codage dans la machine.
Exemple de codage en plusieurs phases :

- En biologie : de l'ADN à la protéine.
- En informatique : du son au fichier mp3

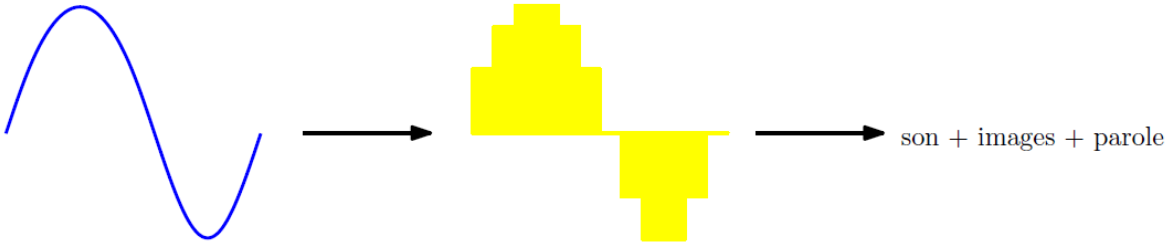
Et maintenant ?

- En biologie :





- En Multi-média :



Plan

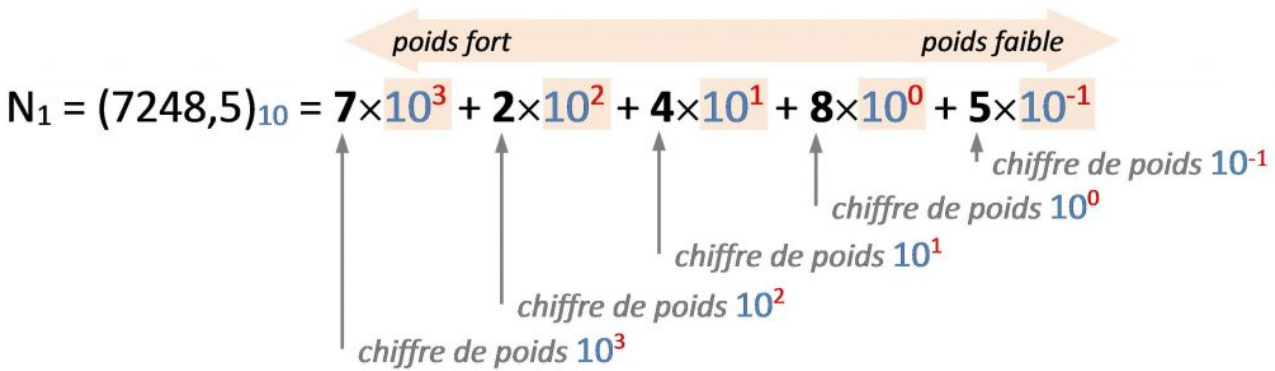
Principe d'une base

- Base : le nombre qui définit le système de numération
- Base du système décimal = 10, base du système octal = 8,. . .

Formule magique en base β

$$\sum_{i=0}^{n}(b_i\beta^i) = b_n\beta^n + \ldots + b_2\beta^2 + b_1\beta^1 + b_0 \text{ où}$$

- bi est le chiffre de la base de rang i
- β_i est la puissance de la base β d'exposant de rang *i*



$$N_2 = (1\,0\,0\,1\,1\,0)_2 = \mathbf{1}\times 2^5 + \mathbf{0}\times 2^4 + \mathbf{0}\times 2^3 + \mathbf{1}\times 2^2 + \mathbf{1}\times 2^1 + \mathbf{0}\times 2^0 = (38)_{10}$$

$$N_3 = (E7B)_{16} = \mathbf{14}\times 16^2 + \mathbf{7}\times 16^1 + \mathbf{11}\times 16^0 = (3707)_{10}$$

Le système décimal

- Origine : le nombre de doigts ?
- 10 digits : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Poids du digit = la puissance de 10 qu'il multiplie Système de numération de position
- Les digits s'écrivent de gauche à droite, par ordre décroissant des puissances de 10.
- La formule magique se précise, pour un nombre x composé de n chiffres dont b_i est un des 10 digits

$$(x)_{10} = \sum_{i=0}^{n} (b_i 10^i) = b_n 10^n + \ldots + b_2 10^2 + b_1 10^1 + b_0$$

$N_2 = (1\,0\,0\,1\,1\,0)_2 = \mathbf{1}\times 2^5 + \mathbf{0}\times 2^4 + \mathbf{0}\times 2^3 + \mathbf{1}\times 2^2 + \mathbf{1}\times 2^1 + \mathbf{0}\times 2^0 = (38)_{10}$

$N_3 = (E7B)_{16} = \mathbf{14}\times 16^2 + \mathbf{7}\times 16^1 + \mathbf{11}\times 16^0 = (3707)_{10}$

Le système octal

- Origine : pratique car c'est une puissance de 2
- 8 digits : 0, 1, 2, 3, 4, 5, 6, 7
- On adapte la formule magique :

$$(x)_8 = \sum_{i=0}^{n}(b_i 8^i) = b_n 8^n + \ldots + b_2 8^2 + b_1 8^1 + b_0$$

- Pour lever les ambiguités : $(x)_\beta$ pour préciser le système de numération

$$(2011)_{10} = (3 \times 8^3) + 78^2 + 38^1 + 3 = (3733)_8$$

Le système binaire

- 2 digits : 0 et 1 (vrai et Faux, ON et OFF, Oui et Non . . .)
- On adapte la formule magique :

$$(x)_2 = \sum_{i=0}^{n} (b_i 2^i) = b_n 2^n + \ldots + b_2 2^2 + b_1 2^1 + b_0$$

$$(2009)_{10} = 1024+512+256+128+64+0\times32+16+8+0\times4+0\times2+1$$

$$\Rightarrow (2009)_{10} = (11111011001)_2$$

- Binary digits = Bits la plus petite unité
- Un octet = 8 bits (et en anglais : un byte)
- Un octet = la taille nécessaire pour coder en binaire un caractère parmi 256 ($256 = 2^8$)

Préfixe binaire (kilo, mega . . .)

- Souvent utilisés lorsqu'on a affaire à de grandes quantités d'octets : puissances de 2
- Ne pas confondre 15 Mbit et 15 Mo = 15 Mbytes . . .
- Dérivés (mais différents) des préfixes SI (Système International d'Unités : puissance de 10)

Nom	Symbole	Puissance	∽Déc	Nombre
unité		$2^0 = 1$	10^0	un
kilo	k/K	$2^{10} = 1024$	10^3	mille
mega	M	$2^{20} = 1048576$	10^6	million
giga	G	$2^{30} = 1073741824$	10^9	milliard
tera	T	$2^{40} = 1099511627776$	10^{12}	billion
peta	P	$2^{50} = 1125899906842624$	10^{15}	billiard
exa	E	$2^{60} = 1152921504606846976$	10^{18}	trillion

Exemple de confusion dans les systèmes de mesure

Délinquance des fabricants de disques dur de 1 To

- Informatique : $1024 \times 1024 \times 1024 \times 1024 = 1 T_o$
- Fabricant de disque :$1000 \times 1000 \times 1000 \times 1000 \approx 0.9.95 T_o$

Le système hexadécimal

- Pratique pour adressage mémoire (Unité de RAM = 2^i)
- 16 digits : 0, 1, . . . , 9, A, B, C, D, E, F
- On adapte la formule magique :

$$(x)_{16} = \sum_{i=0}^{n} (b_i 16^i) = b_n 16^n + \ldots + b_2 16^2 + b_1 16^1 + b_0$$

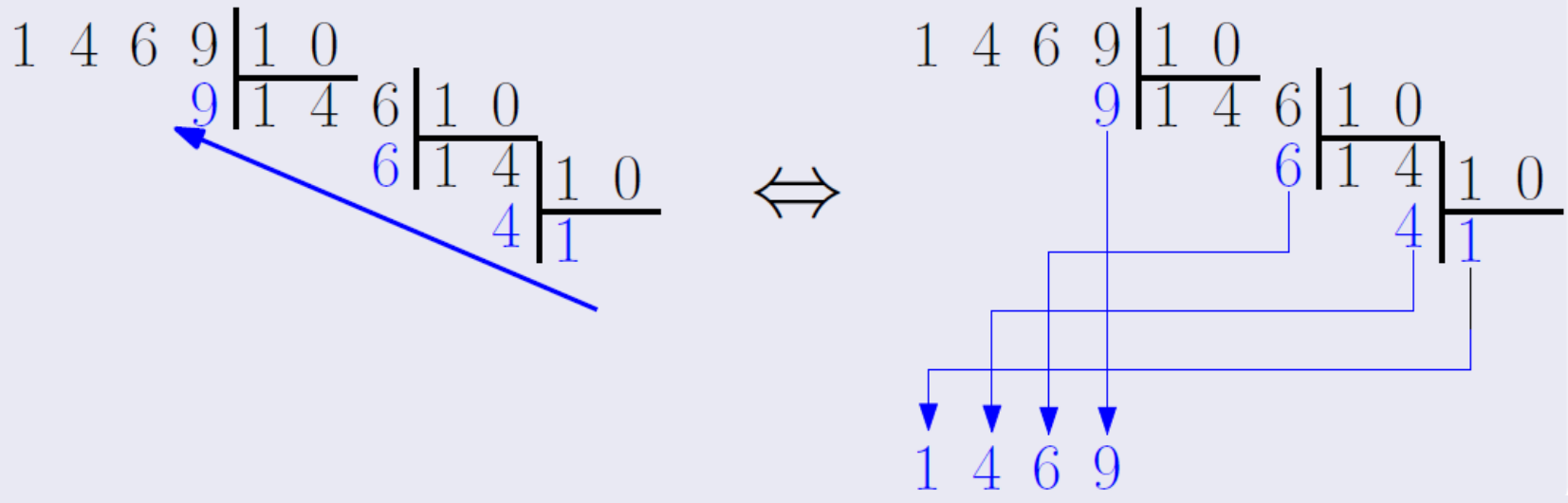
$$(2009)_{10} = 7 \times 16^2 + 13 \times 16 + 9 = 7 \times 16^2 + D \times 16 + 9$$

$$\Rightarrow (2009)_{10} = (7D9)_{16}$$

Changement de base

Changement de base : Principe général

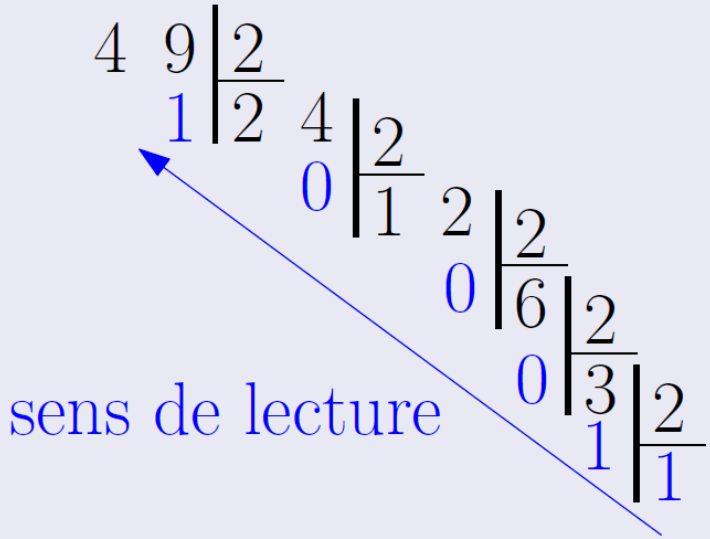
Décomposition en puissances de 10



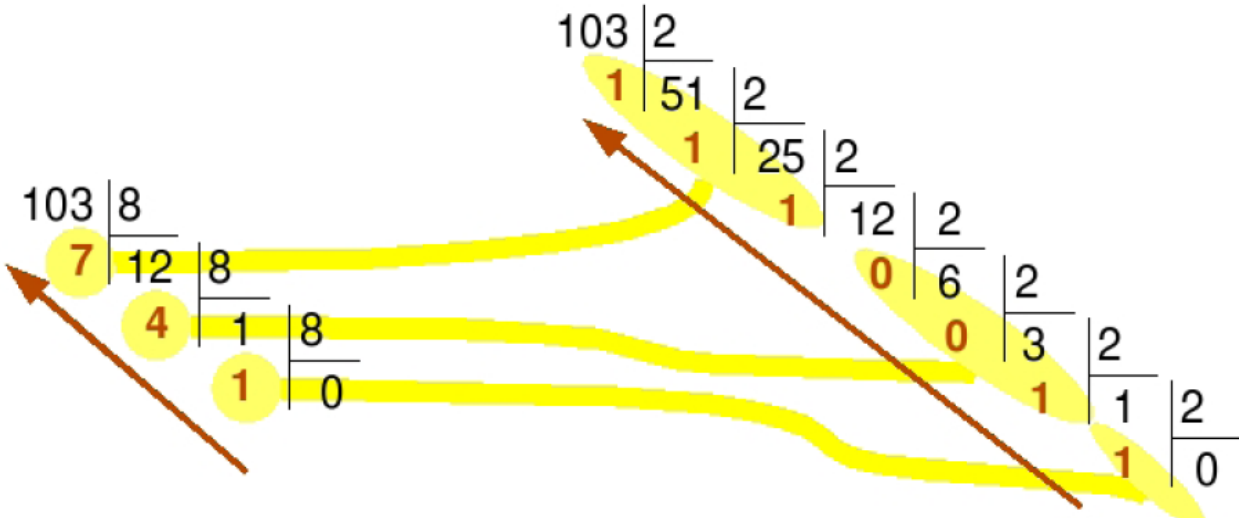
On réalise autant de divisions par 10 que nécessaires, en gardant les restes, et on lit de droite à gauche

$\Rightarrow$ idem pour les autres bases

- Conversion décimal vers binaire : Même principe, divisons successives par 2, on garde les restes
- $(49)_{10} = (110001)_2$



- Relation entre binaire et octal : Examinons $(103)_{10}$ en bases 8 et 2



$$(103)_{10} = (148)_8 = (1100111)_2$$

Relation binaire / puissances de 2

- Equivalence : trois bits $\Longleftrightarrow$ chiffre octal
- Conversion simplifiée
$$\begin{array}{cccc} (& 1 & 4 & 7 \quad)_8 \\ (& \overbrace{001} & \overbrace{100} & \overbrace{111} \quad)_2 \end{array}$$
- Autre exemple :
$$\begin{array}{cccccc} (& 2 & 0 & 5 & 4 &)_8 \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ (& \overbrace{010} & \overbrace{000} & \overbrace{101} & \overbrace{100} &)_2 \end{array}$$
- Même type de relation entre binaire / hexa (groupe de 4 bits)
$$\begin{array}{cccccc} (& A & 5 & 0 & F &)_{16} \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ (& \overbrace{1010} & \overbrace{0101} & \overbrace{0000} & \overbrace{1111} &)_2 \end{array}$$

Relation octal / hexadécimal

Astuce

Il suffit de passer par le binaire :

- De l'octal au binaire :

$$\begin{array}{cccccc} (& 3 & 7 & 1 & 2 &)_8 \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ (& \overbrace{011} & \overbrace{111} & \overbrace{001} & \overbrace{010} &)_2 \end{array}$$

- Du binaire à l'hexadécimal :

$$\begin{array}{ccccc} (& \underbrace{0111} & \underbrace{1100} & \underbrace{1010} &)_2 \\ & \updownarrow & \updownarrow & \updownarrow & \\ (& 7 & C & A &)_{16} \end{array}$$

Quelques propriétés sur la numération

Nombre de chiffres de x dans une base

Nombre de chiffres de x dans une base

Soit $x = b_n b_{n-1} \dots b_2 b_1 b_0$ écrit dans la base β.

- Si $b_n \neq 0$, on note $n+1 = N_\beta(x)$ le nombre de chiffres nécessaires pour exprimer x dans la base β.
- Estimons $N_\beta(x)$

Théorème

$N_\beta(x)$ est le plus petit entier strictement supérieur à $Log_\beta(x)$

Exemple : $\beta = 2$ et $x = (1503)_{10}$

$ln(1503) = 7.32$ et $ln(2) = 0.693 \Rightarrow Log_2(1503) = \frac{7.32}{0.693} = 10.6$
$\Rightarrow N_2(1503) = 11$.
Il faut donc 11 bits pour représenter $(1503)_{10}$ en binaire.
En effet, ce nombre s'écrit, en base 2 : 10111011111

Indifférence de la base β pour x grand

Le rapport $\frac{N_\beta(x)}{N'_\beta(x)}$ tend vers $\frac{Log(\beta(x))}{Log(\beta'(x))}$ quand x tend vers l'infini

Exemple

Pour écrire un grand nombre en base 2, il faut environ 3.32 fois plus de digits qu'en base 10 car $\frac{Log(\beta(10))}{Log(\beta(2))} = 3.32\ldots$

Bienfaits de la représentation binaire

Calculer y^x , si x entier positif

$y^2 = y \times y$ puis $y^3 = y^2 \times y$ puis $y^4 = y^3 \times y$

Méthode plus astucieuse et plus rapide

1. On représente x en binaire :
   $x = b_n 2^n + b_{n-1} 2^{n-1} \ldots + b_1 \times 2 + b_0$
2. Du coup, $y^x = y^{b_n 2^n + b_{n-1} 2^{n-1} * \ldots * b_2 2^2 * b_1 2^1 * b_0}$ Ce qui revient à
   $y^x = (y^{2^n})^{b_n} + (y^{2^{n-1}})^{b_{n-1}} + \ldots + (y^2)^{b_1} + (y)^{b_0}$
3. Or on simplifie ce produit car

$$(y^{2^k})^{b_k} = \begin{cases} y^{2^k} & si \quad b_k = 1 \\ 1 & si \quad b_k = 0 \end{cases}$$

4. On multiplie tous les y^{2^i} pour lesquels b_i n'est pas nul.

Les bienfaits de la représentation binaire

Calculons $y^{1041} = y^1 \times y^{16} \times y^{1024}$

1. Méthode classique :
   On calcule $y^2 = y \times y$ puis $y^3 = y^2 \times y$ puis
   $y^4 = y^3 \times y \ldots y^{60} \ldots y^{780} \ldots y^{1000} \ldots y^{1040}$ et enfin y^{1041}.
   Ce qui fait $x - 1 = 1040$ multiplications
2. Méthode via système binaire :
   On calcule $y^2 = y \times y$ puis $y^4 = y^2 \times y^2$ puis $y^8 = y^4 \times y^4$ puis
   $y^{16} = y^8 \times y^8$ puis y^{32} puis y^{64} puis y^{128} puis y^{256} puis y^{512} puis
   $y^{1024} = y^{512} \times y^{512}$
   On calcule pour finir $y^1 \times y^{16} \times y^{1024}$.
   Ce qui fait au final $10 + 2$ multiplications

Représentation des nombres

De l'infini au fini

Les nombres en mathématiques

Une infinité d'entiers naturels ($\mathbb{N}$), une infinité de réels ($\mathbb{R}$), et la précision des irrationnels

Les nombres en informatique

- Représentation dans le système binaire
- Limites matérielles
  - Quelle que soit la taille d'une disquette (sa capacité), il y aura toujours un entier qui ne pourra pas y être stocké. Soit une disquette de taille $3kBits = 3 \times 1024 = 3072$ chiffres binaires : pas d'entier $\geqslant 2^{3072}$.
  - Représentation approchée des réels
    Pas de représentation numérique exacte de $\sqrt{2}$: nécessité de représentations symboliques

Entiers non-signés

- Les entiers naturels (zéro et les positifs)
- Pas de gestion du signe : codage binaire pur
- Sur un octet, on représente les entiers de 0 à $2^8 - 1$
- Sur 2 octets, on représente les entiers de 0 à $2^{16} - 1$

Entiers signés

- Tous les entiers (dans la limite de capacité)
- Il faut un bit pour représenter le signe (positif ou négatif) et codage binaire de la valeur absolue
- Sur 1 octet, on représente les entiers de $-2^7 + 1$ à $2^7 - 1$
- Sur 2 octets, on représente les entiers de $-2^{15} + 1$ à $2^{15} - 1$

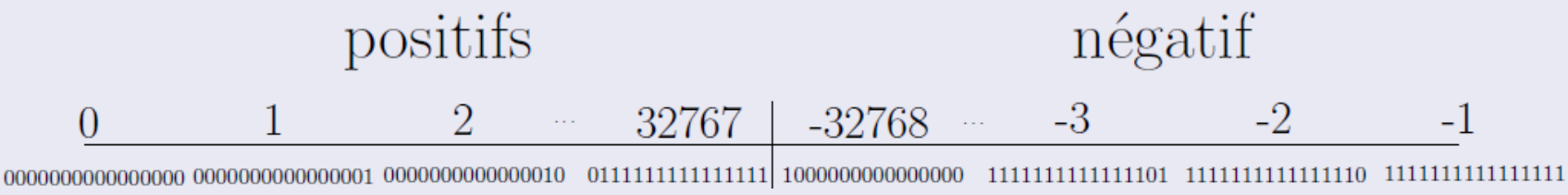
Représentation binaire des données entières

Alternative : le complément à deux

- Toujours éventuellement un bit pour le signe, mais autre façon de coder la valeur absolue
  1. On exprime la valeur en base 2
  2. Tous les bits sont inversés
  3. On ajoute une unité au résultat

- 1 = (0000000000000001)
- -1 = (1111111111111111)
- 3 = (0000000000000011)
- -3 = (1111111111111101)
- 32767 = (0111111111111111)
- -32767 = (1000000000000001)

Répartition des entiers dans le complément à deux



Pourquoi utiliser cette représentation ?

Pour simplifier l'addition !

$$\begin{array}{rr} 0000000000000011 & 3 \\ +1111111111111111 & +(-1) \\ \hline 10000000000000010 & 2 \end{array}$$

Représentation des réels

Une approximation

- Problématiques :
  1. Assurer la précision derrière la virgule
  2. Pouvoir représenter de grands nombres
     152140021536955471089444875,000000000000000000000001
- Avant et après la virgule

Solutions classiques

- Virgule fixe
- Virgule flottante

Virgule fixe

Nombre fixe de chiffres après la virgule

- Opérations plus simple → processeurs moins chers
- Plus facile à coder dans la machine

Deux parties

- La partie entière codée en binaire (complément à deux)
- La partie décimale : chaque bit correspond à l'inverse d'une puissance de 2

Deux parties

$$-3,62510 = \underbrace{1111111111111101}_{-3} \quad \underbrace{1010000000000000}_{0.625 = 0,5 + 0,125 = \frac{1}{2} + \frac{0}{4} + \frac{1}{8}}$$

Capacité de représentation en virgule fixe

Représentation rigide

- Petits nombres : gaspillage des digits à gauche de la virgule
- Peu de décimales : gaspillage à droite
- Plus simple à mettre en œuvre, pour des ordres de grandeur comparables

Bornes

Si n est le nombre de bits de la partie entière, et d est le nombre de bits de la partie fractionnaire

- Borne maximale : $2^{n-1} - \frac{1}{2d}$
- Borne minimale : -2^{n-1}

Exemple : Partie décimale de 0,347 en binaire

$0,347 \times 2 = 0,694 < 1 \Rightarrow$ on pose 0 : $0,347 = 0,0$
$0,694 \times 2 = 1,388 > 1 \Rightarrow$ on pose 4 : $0,347 = 0,01$
$0,388 \times 2 = 0,766 < 1 \Rightarrow$ on pose 0 : $0,347 = 0,010$
$0,766 \times 2 = 1,552 > 1 \Rightarrow$ on pose 4 : $0,347 = 0,0101$
$0,552 \times 2 = 1,104 > 1 \Rightarrow$ on pose 4 : $0,347 = 0,01011$
$0,104 \times 2 = 0,208 < 1 \Rightarrow$ on pose 0 : $0,347 = 0,010110$
$0,208 \times 2 = 0,416 < 1 \Rightarrow$ on pose 0 : $0,347 = 0,0101100$
$0,416 \times 2 = 0,832 < 1 \Rightarrow$ on pose 0 : $0,347 = 0,01011000$
$0,832 \times 2 = 1,664 > 1 \Rightarrow$ on pose 4 : $0,347 = 0,010110001$
$0,664 \times 2 = 1,328 > 1 \Rightarrow$ on pose 4 : $0,347 = 0,0101100011$

Virgule flottante

Solution la plus répandue

- Norme IEEE 75 : deux formats (32bits et 64bits) selon précision (simple et double)
- Ordinateurs actuels : implémentation matérielle de ce mode de représentation (dans le micro-processeur)

Un triplet

Le signe s - La mantisse m - L'exposant e : $x = sm\beta^e$

$$-1540,412654 = -1,540412654.10^3 = -0,1540412654.10^4$$

$$101110,001101 = 1,01110001101.2^5$$

Virgule flottante

Remarques

- Mantisse de taille fixée
- On fait flotter la virgule en faisant varier e
- Base souvent 2 (Hexa chez anciennes machines, 10 chez certaines calculatrices)

Précisions $1 \leqslant m < 2$

	taille	**signe**	**e**	**m**	**valeur**
Simple précision	32b	1	8	23	$-1^s m 2^{e-127}$
Double précision	64b	1	11	52	$-1^s m 2^{e-1023}$

Exemple du nombre -6,62510

En simple précision

- Binaire (valeur absolue) : 110,101
- Normalisation de la mantisse : $1,10101.2^2$
- Occupation des 24 bits de mantisse
  $1,\underbrace{10101000000000000000}$
- Décalage de l'exposant (En S.P. Biais = 127) donc
  exposant $= (2 + 127)_{10} = (10000001)_2$
- Signe = 1 car négatif

1	10000001	10101000000000000000000

Approximations et erreurs

Un exemple pour justifier...

Chez les entiers

- Supposons les entiers positifs codés sur 2 octets (de 0 à 65535)
- Calculons $(39001 + 27446)_{10} = 66447_{10}$

$$\begin{array}{r} 1001100001011001 \\ +0110101100110110 \\ \hline = 1\underbrace{0000001110001111}_{=(911)_{10}} \end{array}$$

- $\Rightarrow$ Dépassement de capacité

Un autre exemple pour justifier...

Chez les réels (flottants)

- Considérons la représentation en virgule flottante D.P.
- Le nombre $2^{60} + 1$ n'est pas représentable (pas assez de capacité), et approximé par 2^{60}

$$1,0000000000000000\ldots000001$$

- Conséquence : $(2^{60} + 1) - 2^{60} = 0$
- $\Rightarrow$ Epsilon de la machine

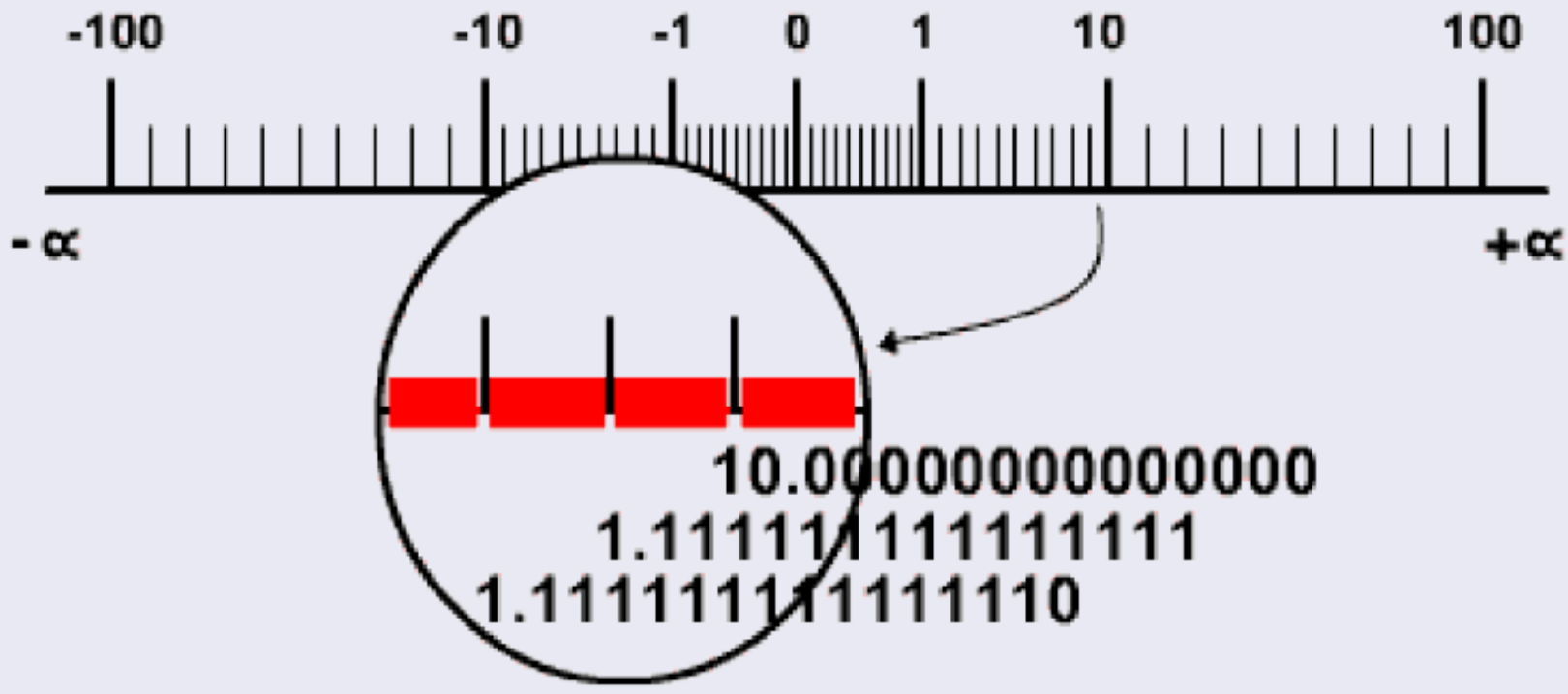
Dispositions particulières (exposant)

- bit d'information : NaN : Not a Number (avec propagation)
- bit d'information : +INF et -INF (avec propagation)

Intervalle et précision

Infiniment petit et infiniment grand

Densité des réels (ici codés en 16 bits)



Ecart entre x et sa représentation

Estimation de l'approximation

$$\frac{\Delta x}{x} = \frac{\Delta m}{m} \leqslant \frac{\beta^{-N}}{\beta^{-1}} = \beta^{1-N}$$

Exemple en base 2

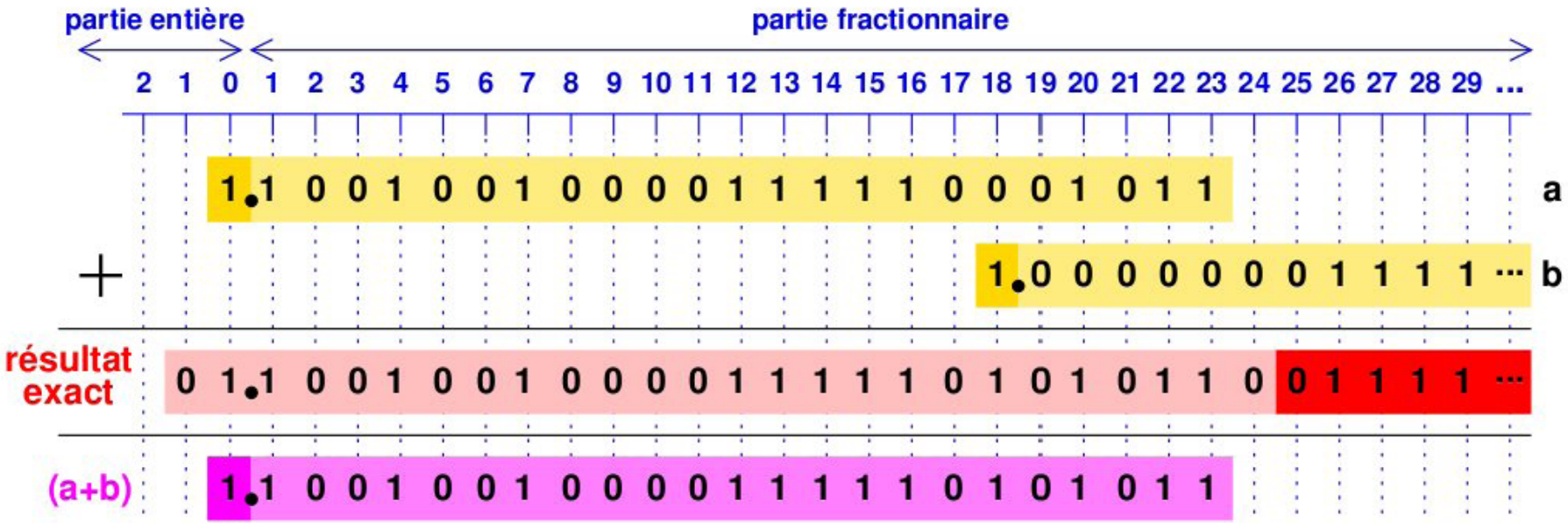
- En simple précision, mantisse sur 23 bits

$$\frac{\Delta x}{x} \leqslant 2^{-22} = 0,000000238 \approx 2.38.10^{-7}$$

- En double précision, mantisse sur 52 bits

$$\frac{\Delta x}{x} \leqslant 2^{-51} \ll 1$$

Illustration du phénomène d'absorption



Conséquence de ces approximations

Erreurs d'arrondis

Lorsque N est dépassé, approximation :

- arrondi vers la décimale la plus proche
- troncature

$\Rightarrow$ Calcul flottant non associatif

$$\sum_{i=1}^{n} \frac{1}{i} \neq \sum_{i=n}^{1} \frac{1}{i}$$

Somme des inverses : Leçon

Pour $n = 10^9$

Lorsque N est dépassé, approximation :

- Simple précision 32 bits :
  - $\sum_{i=1}^{n} \frac{1}{i} = 15.403$
  - $\sum_{i=n}^{1} \frac{1}{i} = 18.807$
- Double précision (64 bits)
  - $\sum_{i=1}^{n} \frac{1}{i} = 21.300481502385$
  - $\sum_{i=n}^{1} \frac{1}{i} = 21.300481502361$

Règle générale

Sommer en premier les termes ayant la plus petite valeur absolue

Phénomène de compensation (élimination)

Erreurs d'approximation liées à la soustraction

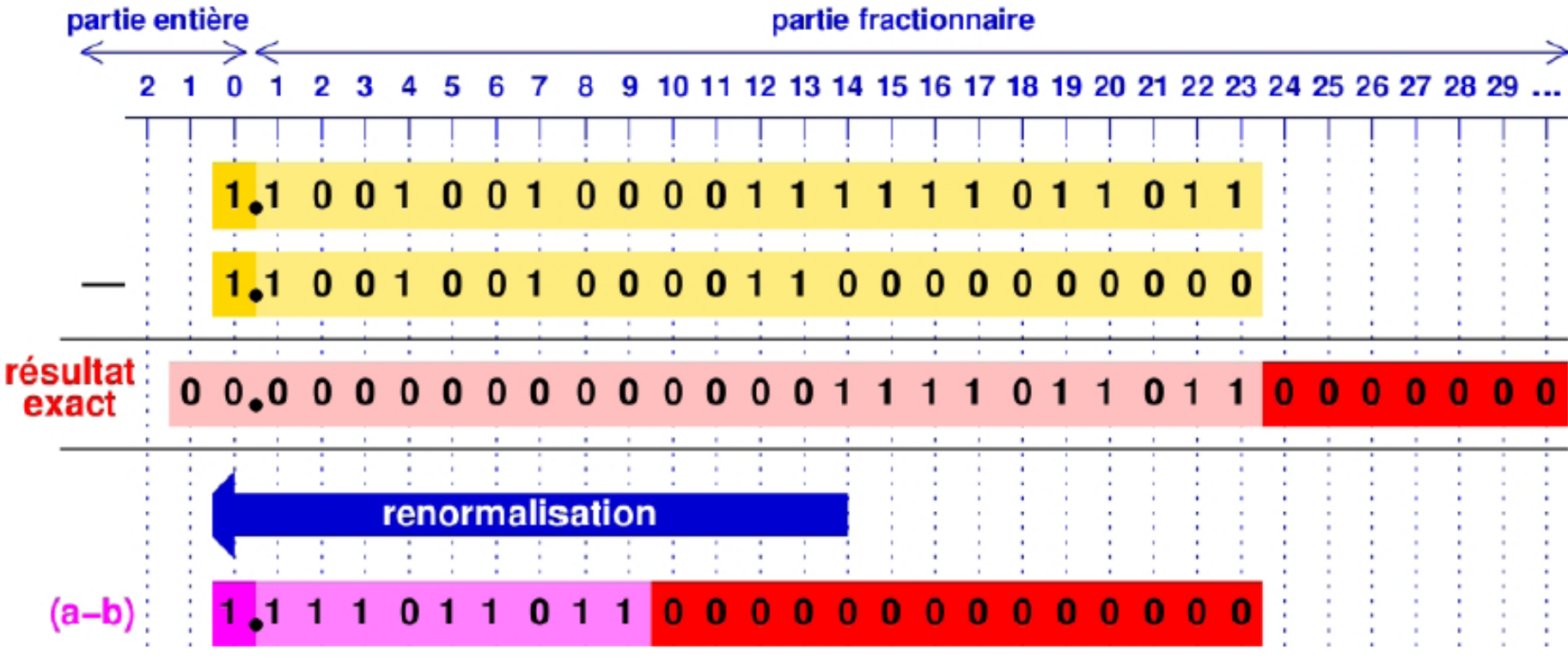
Les termes sommés s'annulent si trop proches

$$e^{-10} \approx \sum_{i=1}^{n} (-1)^k \frac{10^k}{k!} \Rightarrow e^{-10} \approx -1.295050418187$$

Eviter les sommations dans lesquelles les termes de signes opposés se compensent

$$e^{10} \approx \sum_{i=1}^{n} \frac{10^k}{k!} \Rightarrow e^{-10} = \frac{1}{e^{10}} \approx 0.000045401$$

Visualisation de l'élimination



Coefficient d'amplification

- Deux façons de calculer

$$I_n = \int_0^n x^n e^{-x} dx = -e^{-1} + nI_{n-1} \text{ et } I_0 = 1 - e^{-1}$$

qui converge vers 10^3

1. En montant (de I_0 à I_n)
2. En descendant, de $I_{4n} = a$ quelconque à I_n

- Bizarrement, c'est la seconde solution qui converge
- Coefficient d'amplification de l'erreur d'arrondi

0.000001 au lieu de 0.00000141269

Après multiplication par 1000

0.010000 au lieu de 0.0141269

$\Rightarrow$ mieux vaut diviser . . .

Missiles Patriot : 0,34s de retard et 28 morts

En février 1991, guerre du Golfe : un missile Patriot (tueur de missiles Scud) tue 28 soldats

L'erreur : accumulation d'arrondis

- Nombres en virgule fixe sur 24 bits
- Temps compté par horloge interne en 1/10 de seconde

$$1/10 = (0.1)_{10} = (0.0001100110011001100110011001100110011\ldots)_2$$

- Arrondi à 24 chiffres : erreur tous les 10^{mes} de sec.
- Au moment de l'attaque, batterie du Patriote démarrée depuis 100h donc accumulation de 0,34s d'erreur
- Vitesse Scud $\approx 1 km/s$: cible ratée

Explosion d'Ariane 5

Juin 1996, explosion d'Ariane 5 (1 milliard de dollars) Conversion et

Conversion et approximation sévère

- Reprise à l'identique du logiciel d'Ariane 4 pour la gestion des centrales de guidage : nombres de 16 bits en entiers signés
- Tout le reste d'Ariane 5 : nombres de 64 bits en virgule flottante
- Passage d'un système à l'autre de la vitesse horizontale de la fusée par rapport à la plate-forme de tir : plus grande que 32767
- Effet modulo et déviation de la trajectoire impossible à rectifier

Autres bugs célèbres

Pentium d'Intel, 1994

Erreur dans la table de référence des divisions

Conversion Euro

Erreurs d'arrondis, amplifiées par opérations de conversion et de reconversion avant totalisation !

Augmentation du Vancouver Stock Exchange (1983)

(alors que les prix n'ont pas varié) Troncature de 4 vers 3 décimales et amplification quotidienne